



МИНОБРНАУКИ РОССИИ  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«МИРЭА – Российский технологический университет»  
**РТУ МИРЭА**

---

Институт информационных технологий (ИИТ)  
Кафедра корпоративных информационных систем (КИС)

**ОТЧЁТ ПО УЧЕБНОЙ ПРАКТИКЕ**  
**Ознакомительная практика**

приказ Университета о направлении на практику от 09 февраля 2023 г. № 663-С

Отчет представлен к  
рассмотрению:  
Студент группы ИКБО-29-22


« 7 » июня 2023

  
Азиз М.  
(подпись и расшифровка подписи)

Отчет утвержден.  
Допущен к защите:

Руководитель практики  
от кафедры

« 7 » июня 2023

  
Мисаилиди А.А.  
(подпись и расшифровка подписи)

Москва 2023 г.



МИНОБРНАУКИ РОССИИ  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«МИРЭА – Российский технологический университет»  
**РТУ МИРЭА**

---

Институт информационных технологий (ИИТ)  
Кафедра корпоративных информационных систем (КИС)

**ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ НА УЧЕБНУЮ ПРАКТИКУ**  
**Ознакомительная практика**

**Студенту 1 курса учебной группы ИКБО-29-22**

**Азизу Матиулле**

**Место и время практики:** РТУ МИРЭА кафедра КИС, с 09 февраля 2023 г. по 31 мая 2023 г.

**Должность на практике:** студент

**1. СОДЕРЖАНИЕ ПРАКТИКИ:**

1.1. Изучить: методические материалы по курсу, программную документацию языка программирования и стандартных библиотек, API применяемых сервисов.

1.2. Практически выполнить:

- анализ выданных в курсе практических задач;
- поиск, интерпретацию, анализ и ранжирование информации из изученных источников и баз данных, необходимых для решения практических задач;
- решение задач по темам: работа с коллекциями, работа со строками, работа с файлами, основы ООП, стандартные библиотеки, графический интерфейс, использование сторонних API;
- представление результатов выполнения практических задач в требуемом формате (условие, алгоритм, решение задачи, тестирование).

1.3. Ознакомиться: со средствами социального взаимодействия и командной работы в профессиональной среде, с учебно-методическим пособием по ознакомительной практике.

**2. ДОПОЛНИТЕЛЬНОЕ ЗАДАНИЕ:** нет

**3. ОРГАНИЗАЦИОННО-МЕТОДИЧЕСКИЕ УКАЗАНИЯ:** Положение о практической подготовке обучающихся, осваивающих основные профессиональные образовательные программы ВО; учебно-методическое пособие по ознакомительной практике.

Руководитель практики от  
кафедры

Подпись

(Мисаилиди А.А.)

«09» февраля 2023 г.

Задание получил

Подпись

(Азиз М.)

«09» февраля 2023 г.

**СОГЛАСОВАНО:**

Заведующий кафедрой:

Подпись

(Андрианова Е.Г.)

«09» февраля 2023 г.

**Проведенные инструктажи:**

Охрана труда:

Инструктирующий

  
Подпись

«09» февраля 2023 г.

Трохаченкова Н.Н., старший преподаватель кафедры КИС

Инструктируемый

  
Подпись

Азиз М.

Техника безопасности:

Инструктирующий

  
Подпись

«09» февраля 2023 г.

Трохаченкова Н.Н., старший преподаватель кафедры КИС

Инструктируемый

  
Подпись

Азиз М.

Пожарная безопасность:

Инструктирующий

  
Подпись

«09» февраля 2023 г.

Трохаченкова Н.Н., старший преподаватель кафедры КИС

Инструктируемый

  
Подпись

Азиз М.

С правилами внутреннего распорядка ознакомлен:

  
Подпись

«09» февраля 2023 г.

Азиз М.





МИНОБРНАУКИ РОССИИ  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«МИРЭА – Российский технологический университет»  
**РТУ МИРЭА**

**РАБОЧИЙ ГРАФИК ПРОВЕДЕНИЯ  
ОЗНАКОМИТЕЛЬНОЙ ПРАКТИКИ**

студента Азиза М. 1 курса группы ИКБО-29-22 очной формы обучения, обучающегося по направлению подготовки 09.03.04 Программная инженерия

Неделя	Сроки выполнения	Этап	Отметка о выполнении
1	09.02 – 15.02	Подготовительный этап, включающий в себя организационное собрание (Вводная лекция о порядке организации и прохождения производственной практики, инструктаж по технике безопасности, получение задания на практику)	
2-3	17.02 – 01.03	Выполнение заданий по теме «Основные конструкции языка, коллекции» (анализ задач; поиск информации для решения; решение задач; представление результатов в требуемом формате)	
4-5	02.03 – 14.03	Выполнение заданий по теме «Строки, работа с файлами» (анализ задач; поиск информации для решения; решение задач; представление результатов в требуемом формате)	
6-7	16.03 – 28.03	Выполнение заданий по теме «Основы ООП» (анализ задачи; поиск информации для решения; решение задачи; представление результатов в требуемом формате)	
8-9	30.03 – 11.04	Выполнение заданий по теме «Стандартные библиотеки языка программирования» (анализ задач; поиск информации для решения; решение задач; представление результатов в требуемом формате)	
10-11	13.04 – 25.04	Выполнение заданий по теме «Графический интерфейс и внешние библиотеки» (анализ задачи; поиск информации для решения; решение задачи; представление результатов в требуемом формате)	
12-14	27.04 – 16.05	Использование сторонних API для создания приложений, (анализ задачи; поиск информации для решения; решение задачи; представление результатов в требуемом формате)	
15-16	18.05 – 31.05	Оформление материалов отчета в полном соответствии с требованиями на оформление письменных учебных работ студентов	

Руководитель практики от кафедры

/Мисаилиди А.А. /

Обучающийся

/ Азиз М. /

Согласовано:

Заведующий кафедрой

/Андрианова Е.Г., к.т.н., доцент/

## **Оглавление**

<b>1) Тема «Основные конструкции языка, коллекции» .....</b>	<b>6</b>
<b>2) Тема «Строки, работа с файлами» .....</b>	<b>14</b>
<b>3) Тема «Основы объектно-ориентированного программирования».....</b>	<b>18</b>
<b>4) Тема «Стандартные библиотеки языка программирования» .....</b>	<b>26</b>
<b>5) Тема «Графический интерфейс и внешние библиотеки» .....</b>	<b>37</b>
<b>6) Использование сторонних API для создания приложений .....</b>	<b>46</b>
<b>7) Заключение.....</b>	<b>46</b>

## 1) Тема «Основные конструкции языка, коллекции»

### Задача 1

Условие задачи: Написать функцию `very_even(number)`, которая определяет является ли число "очень четным". Однозначное число "очень четное", если оно четное. Числа больше 10 "очень четные", если сумма их цифр - "очень четное" число.

### Решение задачи:

```
def very_even(number):
    if number < 10 :
        return number % 2 == 0
    digit = [int(i) for i in str(number)]
    sums = sum (digit)
    return very_even(sums)
```

### Тестирование:

№	Тест	Ожидаемое значение	Полученное значение
1	4	True	True
2	5	False	False
3	12	False	False
4	1234	False	False
5	7897	True	True

### Задача 2

Условие задачи: Задан список, состоящий из не менее трех целых чисел. Список содержит или все четные числа кроме одного или все нечетные числа кроме одного. Написать функцию `find_outlier`, которая возвращает число-исключение

### Решение задачи:

```
def find_outlier(integers):
    even = []
    odd = []
    for i in range(len(integers)):
        if integers[i] % 2 == 0 :
            even.append(integers[i])
        else :
            odd.append(integers[i])
    if len(even) == 1 : return even[0]
    elif len(odd) ==1 : return odd[0]
```

Тестирование:

№	Тест	Ожидаемое значение	Полученное значение
1	[2, 4, 6, 8, 10, 3]	3	3
2	[2, 4, 0, 100, 4, 11, 2602, 36]	11	11
3	[160, 3, 1719, 19, 11, 13, -21]	160	160

**Задача 3**

Условие задачи: GPS в машине каждые s секунд записывает в список x значение пройденного расстояния с начала пути. Написать функцию `gps(s, x)`, которая возвращает максимальную среднюю скорость среди интервалов.

$\text{Средняя\_скорость} = 3600 * \text{расстояние} / s$

Решение задачи:

```
def gps(s, x):
    if len(x) < 2 :
        speed = x[0] / s;
        return int (speed)
    speeds =[]
    for i in range (1, len(x)) :
        avespeed = (x[i]-x[i-1]) / (s / 3600)
        speeds.append(avespeed)
    return int(max(speeds))
```

Тестирование:

№	Тест	Ожидаемое значение	Полученное значение
1	12, [0.0, 0.24, 0.48, 0.72, 0.96, 1.2, 1.44, 1.68, 1.92, 2.16, 2.4]	72	72
2	17, [0.0, 0.02, 0.44, 0.66, 0.88, 1.1, 1.32, 1.54, 1.76]	88	88
3	18, [0.0]	0	0

#### Задача 4

Условие задачи: Написать функцию `sameness_index`, которая возвращает индекс N, где сумма целых чисел слева от N равна сумме целых чисел справа от N. Если нет индекса, который бы это сделал, верните -1.

Решение задачи:

```
def sameness_index(arr):  
    for i in range(len(arr)):  
        left = sum(arr[:i])  
        right = sum(arr[i+1:])  
        if left == right: return i  
    return -1
```

Тестирование:

№	Тест	Ожидаемое значение	Полученное значение
1	[1, 2, 3, 4, 5, 6]	-1	-1
2	[20, 10, 30, 10, 10, 15, 35]	3	3
3	[20, 10, -80, 10, 10, 15, 35]	0	0
4	[10, -80, 10, 10, 15, 35, 20]	6	6

#### Задача 5

Условие задачи: Создать список (автосалон), состоящий из словарей (машина). Словари должны содержать как минимум 5 полей (например, номер, модель, год выпуска, ...). В список добавить хотя бы 10 словарей.

Конструкция вида: `cars = [{"id":123456, "model":"Mercedes-Benz", "year": 2019, ...} , {...}, {...}, ...]`.

Реализовать функции:

- вывода информации о всех машинах;
- вывода информации о машине по введенному с клавиатуры номеру;
- вывода количества машин, моложе введенного года;
- обновлении всей информации о машине по введенному номеру;



– удалении машины по номеру.

### Решение задачи:

```
cars = [{"id":1, "model":"Mercedes-Benz", "year": 2019, "colour": "red", "price": 100},
        {"id":2, "model":"Toyota", "year": 2011, "colour": "dark-red", "price": 150},
        {"id":3, "model":"bmw", "year": 2023, "colour": "white", "price": 130},
        {"id":4, "model":"ford", "year": 2010, "colour": "black", "price": 50},
        {"id":5, "model":"Honda", "year": 2012, "colour": "yellow", "price": 70},
        {"id":6, "model":"ferari", "year": 2021, "colour": "golden", "price": 170},
        {"id":7, "model":"chevrolet", "year": 2015, "colour": "pink", "price": 180},
        {"id":8, "model":"Mazda", "year": 2009, "colour": "white", "price": 80},
        {"id":9, "model":"audi", "year": 2002, "colour": "blue", "price": 102},
        {"id":10, "model":"lexus", "year": 2000, "colour": "red", "price": 200},
        ]

def information(cars):
    for car in cars:
        print (car)

def information_by_id(cars):
    print ("Please enter the id of the car you want to search")
    id = int (input ())
    for car in cars:
        if car["id"] == id:
            print (car)

def young_Cars(cars):
    print ("enter the year: ")
    year = int (input ())
    # young_car = max([car['year'] for car in cars])
    young_car_num= [car for car in cars if car['year'] > year]
    # young_car_num= ([car for car in cars if car['year'] == young_car] )# if you
    wanted to see how many put the len and use for loop to print all of them

    print (young_car_num)

def edit_info(cars):
    print ('Enter the id number of the car')
    id = int (input())
    for car in cars :
        if car['id'] == id:
            print ('Enter new model: ')
            car['model'] = str(input())
            print ('\n Enter new year: ')
            car ['year'] = int(input())
            print ('\n Enter new color: ')
            car ['color'] = str(input())
            print ('\n Enter new Price: ')
            car ['price'] = int(input())
    for car in cars:
        print (car)

def delete_car(cars):
    print ("Enter the id of car you want to delete: \n")
    id = int(input())
    for car in cars:
        if car['id'] == id:
            cars.remove(car)
            for car in cars:
```

```

        print (car)

while True:

    print ('choose an option to continue \n 1: Information about all the cars\n 2:
Information about cars by id\n 3: find the most recent cars \n 4: edit information
about a car \n 5: delete a car ')
    user = int (input ())
    if user == 1:
        information(cars)
    elif user == 2:
        information_by_id(cars)
    elif user == 3:
        young_Cars(cars)
    elif user == 4:
        edit_info(cars)
    elif user == 5:
        delete_car(cars)
    else:
        print ("Enter a valid input! \n")

```

### Тестирование:

№	Тест	Ожидаемое значение	Полученное значение
1	1	{"id": 1, "model": "Mercedes-Benz", "year": 2019, "colour": "red", "price": 100} {"id": 2, "model": "Toyota", "year": 2011, "colour": "dark-red", "price": 150} {"id": 3, "model": "bmw", "year": 2023, "colour": "white", "price": 130} {"id": 4, "model": "ford", "year": 2010, "colour": "black", "price": 50}	{"id": 1, "model": "Mercedes-Benz", "year": 2019, "colour": "red", "price": 100} {"id": 2, "model": "Toyota", "year": 2011, "colour": "dark-red", "price": 150} {"id": 3, "model": "bmw", "year": 2023, "colour": "white", "price": 130} {"id": 4, "model": "ford", "year": 2010, "colour": "black", "price": 50}

		{"id": 5, "model": "Honda", "year": 2012, "colour": "yellow", "price": 70} {"id": 6, "model": "ferari", "year": 2021, "colour": "golden", "price": 170} {"id": 7, "model": "chevrolet", "year": 2015, "colour": "pink", "price": 180} {"id": 8, "model": "Mazda", "year": 2009, "colour": "white", "price": 80} {"id": 9, "model": "audi", "year": 2002, "colour": "blue", "price": 102} {"id": 10, "model": "lexus", "year": 2000, "colour": "red", "price": 200}	{"id": 5, "model": "Honda", "year": 2012, "colour": "yellow", "price": 70} {"id": 6, "model": "ferari", "year": 2021, "colour": "golden", "price": 170} {"id": 7, "model": "chevrolet", "year": 2015, "colour": "pink", "price": 180} {"id": 8, "model": "Mazda", "year": 2009, "colour": "white", "price": 80} {"id": 9, "model": "audi", "year": 2002, "colour": "blue", "price": 102} {"id": 10, "model": "lexus", "year": 2000, "colour": "red", "price": 200}
2	2 3	{"id": 3, "model": "bmw", "year": 2023, "colour": "white", "price": 130}	{"id": 3, "model": "bmw", "year": 2023, "colour": "white", "price": 130}
3	3 2015	{"id": 1, "model": "Mercedes-Benz",	{"id": 1, "model": "Mercedes-Benz",

		"year": 2019, "colour": "red", "price": 100} { "id": 3, "model": "bmw", "year": 2023, "colour": "white", "price": 130} { "id": 6, "model": "ferari", "year": 2021, "colour": "golden", "price": 170}	"year": 2019, "colour": "red", "price": 100} { "id": 3, "model": "bmw", "year": 2023, "colour": "white", "price": 130} { "id": 6, "model": "ferari", "year": 2021, "colour": "golden", "price": 170}
4	4 5 Honda Civic 2014 silver 80	{ "id": 5, "model": "Honda Civic", "year": 2014, "colour": "silver", "price": 80}	{ "id": 5, "model": "Honda Civic", "year": 2014, "colour": "silver", "price": 80}
5	5 2	{ "id": 1, "model": "Mercedes-Benz", "year": 2019, "colour": "red", "price": 100} { "id": 3, "model": "bmw", "year": 2023, "colour": "white", "price": 130} { "id": 4, "model": "ford", "year": 2010, "colour": "black", "price": 50} { "id": 5, "model": "Honda", "year": 2012, "colour": "yellow", "price": 70}	{ "id": 1, "model": "Mercedes-Benz", "year": 2019, "colour": "red", "price": 100} { "id": 3, "model": "bmw", "year": 2023, "colour": "white", "price": 130} { "id": 4, "model": "ford", "year": 2010, "colour": "black", "price": 50} { "id": 5, "model": "Honda", "year": 2012, "colour": "yellow", "price": 70}

		{"id": 6, "model": "ferari", "year": 2021, "colour": "golden", "price": 170} {"id": 7, "model": "chevrolet", "year": 2015, "colour": "pink", "price": 180} {"id": 8, "model": "Mazda", "year": 2009, "colour": "white", "price": 80} {"id": 9, "model": "audi", "year": 2002, "colour": "blue", "price": 102} {"id": 10, "model": "lexus", "year": 2000, "colour": "red", "price": 200}	{"id": 6, "model": "ferari", "year": 2021, "colour": "golden", "price": 170} {"id": 7, "model": "chevrolet", "year": 2015, "colour": "pink", "price": 180} {"id": 8, "model": "Mazda", "year": 2009, "colour": "white", "price": 80} {"id": 9, "model": "audi", "year": 2002, "colour": "blue", "price": 102} {"id": 10, "model": "lexus", "year": 2000, "colour": "red", "price": 200}
--	--	--	--

## 2) Тема «Строки, работа с файлами»

### Задача 1

Условие задачи: Написать функцию `olympic_ring`, которая считает количество колец в заданной строке латинских букв (в символе 'a' - одно кольцо, в 'B' - два). Количество колец нужно поделить пополам и округлить в меньшую сторону. Если результат равен 1 или меньше -верните «Not even a medal!», если счет равен 2 - вернуть «Bronze!», если счет равен 3 - «Silver!», если оценка больше 3 - вернуть «Gold!»;

### Решение задачи:

```
def olympic_ring(s):
    ring_count = 0
    for circle in s:
        if circle in 'aAddegqopQROb':
            ring_count += 1
        elif circle == 'B':
            ring_count += 2
    # print (ring_count)
    if ring_count % 2 == 0:
        medal_count = ring_count // 2
    else:
        medal_count = ring_count // 2 + 1
    # print(medal_count)
    if medal_count <= 1:
        return "Not even a medal!"
    elif medal_count == 2:
        return "Bronze!"
    elif medal_count == 3:
        return "Silver!"
    else:
        return "Gold!"
```

### Тестирование:

№	Тест	Ожидаемое значение	Полученное значение
1	wHjMudLwtoPGocnJ	Bronze!	Bronze!
2	eCEHWEPwwnvzMicyaRjk	Bronze!	Bronze!
3	JKniLfLW	Not even a medal!	Not even a medal!
4	EWIZIDFsEIBufsalqof	Silver!	Silver!
5	IMBAWejlGRTDWetPS	Gold!	Gold!



## Задача 2

Условие задачи: На вход поступает список словарей. Написать функцию sentence, которая возвращает строку, состоящую из слов (значений словарей), разделенных пробелом в порядке возрастания целочисленных значений их ключей.

Решение задачи:

```
def sentence(lst):
    result = ""
    pairs = [(int(key), value) for d in lst for key, value in d.items()]
    # print (pairs)
    sorted_pairs = sorted(pairs, key=lambda x: x[0])
    for pair in sorted_pairs:
        result+= (pair[1])+ ' '
    #print (result.strip())
    return result.strip()
```

Тестирование:

№	Тест	Ожидаемое значение	Полученное значение
1	[[{"0": "is"}, {"11": "never"}, {"-100": "Lost"}, {"75": "again"}, {"14": "found"}, {"-9": "time"}]]	Lost time is never found again	Lost time is never found again
2	[[{"100": "overeducated"}, {"0": "never"}, {"15": "or"}, {"11": "overdressed"}, {"-500": "You"}, {"-2": "can"}, {"7": "be"}]]	You can never be overdressed or overeducated	You can never be overdressed or overeducated

## Задача 3

Условие задачи: Даны результаты забегов в формате "h|m|s, h|m|s, h|m|s" (h – часы, m – минуты, s – секунды). Написать функцию stat, которая возвращает строку в формате "Range: hh|mm|ss Average: hh|mm|ss" (range –

разница между максимальным и минимальным значением, average – среднее значение).

#### Решение задачи:

```
def stat(times):
    # Split the times string into a list of individual time strings
    time_list = times.split(", ")
    # Convert each time string to a number of seconds
    time_in_seconds = [int(t.split("|")[0])*3600 + int(t.split("|")[1])*60 +
int(t.split("|")[2]) for t in time_list]
    # Calculate the range and average time in seconds
    time_range = max(time_in_seconds) - min(time_in_seconds)
    time_average = sum(time_in_seconds) / len(time_in_seconds)
    def sectotime(secs):
        hours = secs // 3600
        mins = (secs % 3600) // 60
        secs = secs % 60
        formnum = "{:02d}|{:02d}|{:02d}".format(int(hours), int(mins), int(secs))
        return formnum
    return "Range: "+sectotime(time_range)+" Average: "+sectotime(time_average)
```

#### Тестирование:

№	Тест	Ожидаемое значение	Полученное значение
1	01 15 59, 1 47 16, 01 17 20, 1 32 34, 2 17 17	Range: 01 01 18 Average: 01 38 05	Range: 01 01 18 Average: 01 38 05
2	02 15 59, 2 47 16, 02 17 20, 2 32 34, 2 17 17, 2 22 00, 2 31 41	Range: 00 31 17 Average: 02 26 18	Range: 00 31 17 Average: 02 26 18

#### **Задача 4**

Условие задачи: Создать txt-файл, вставить туда любую англоязычную статью из Википедии.

Реализовать одну функцию, которая выполняет следующие операции:

- прочитать файл построчно;
- непустые строки добавить в список;
- удалить из каждой строки все цифры, знаки препинания, скобки, кавычки и т.д. (остаются латинские буквы и пробелы);

- объединить все строки из списка в одну, используя метод join и пробел, как разделитель;
- создать словарь вида {“слово”: количество, “слово”: количество, ... } для подсчета количества разных слов,
- где ключом будет уникальное слово, а значением - количество;
- вывести в порядке убывания 10 наиболее популярных слов, используя форматирование
- (вывод примерно следующего вида: “ 1 place --- sun --- 15 times \n....”);
- заменить все эти слова в строке на слово “PYTHON”;
- создать новый txt-файл;
- записать строку в файл, разбивая на строки, при этом на каждой строке записывать не более 100 символов при этом не делить слова.

#### Решение задачи:

```
def wiki_function(filename):
    # Read the file line by line and add non-empty lines to a list
    print ('>>Read the file line by line and add non-empty lines to a list')
    with open(filename, "r") as f:
        for line in f:
            print (line)
    print ("\n\n")
    # print ('Select what to do with the text: \n1-> Add non-empty lines into a
    list\n->Remove everything except latins characters\n->')

    print ('>>adding non-empty lines to a list:')
    lines = []
    with open(filename, "r") as f:
        for line in f:
            line = line.strip()
            if line:
                lines.append(line)
    print (lines)

    print ("->Removing other characters except latins: \n")
    latins = ''
    with open(filename, "r") as f:
        for line in f:
            for lat in line:
                if lat.isalpha() or lat.isspace():
                    latins+= lat
    print (latins)
    print ('\n\n->Joining all characters from list using join method with comman as
    separator: ')
    string = " ".join(lines)
    print (string)
    print("\n\n-> Creating a dic which contains the words as key and value as
```

```

quantity: ")
word_count = {}
for word in string.split():
    word = word.lower()
    if word not in word_count:
        word_count[word] = 0
    word_count[word] += 1
print (word_count)
print ("->10 most used words in txt: \n")
sorted_words = sorted(word_count.items(), key=lambda x: x[1], reverse=True)
for i, (word, count) in enumerate(sorted_words[:10]):
    print(f"{i+1} place --- {word} --- {count} times")

print ("->changing them to python: \n")
#!====
top_words = [word for word, count in sorted_words[:10]]
mylist = []
for word in top_words:
    mylist.append(word)
pattern = r'\b(?:{})\b'.format('|'.join(mylist)) #/b checks the boundaries
modified_string = re.sub(pattern, 'Python', string)

#!to write a new file with nomore than 100 chars in a line

print ("->writing the string to a new file: \n\n")
with open("output.txt", "w") as f:
    while len(modified_string) > 100:
        index = modified_string.rfind(" ", 0, 100)
        if index == -1:
            index = 100
        f.write(modified_string[:index] + "\n")
        modified_string = modified_string[index+1:]
    f.write(modified_string)
return 1
# Вызов функции
wiki_function("python.txt")

```

### 3) Тема «Основы объектно-ориентированного программирования»

#### Задача

Условие задачи: Создать класс Person с полями имя, фамилия, возраст.

Добавить конструктор класса.

Создать производный от Person класс Student. Новые поля: номер студенческого билета, зачетная книжка (словарь вида предмет: отметка). Определить конструктор, с вызовом родительского конструктора Определить функции добавления отметки в зачетную книжку, получения отметки по предмету, форматированной печати всей зачетной книжки. Переопределить

метод преобразования в строку для печати основной информации (ФИ, возраст, номер студенческого).

Создать производный от Person класс Professor. Новые поля: номер удостоверения, должность, преподаваемые предметы (список строк). Определить конструктор, с вызовом родительского конструктора. Определить функции изменения должности, добавления и удаления предмета. Переопределить метод преобразования в строку для печати основной информации (ФИ, возраст, номер удостоверения, должность).

Создать класс Group. Поля: номер группы, список группы (список экземпляров класса Student), преподаватель-куратор (экземпляр класса Professor). Определить конструктор. Переопределить метод преобразования в строку для печати всей информации о группе (с использованием переопределения в классах Professor и Student). Переопределить методы получения количества студентов функцией len, получения студента/преподавателя по индексу, изменения по индексу, удаления по индексу (0 индекс - преподаватель, начиная с 1 - студенты). Переопределить операции + и - для добавления или удаления студента из группы. Добавить функцию создания txt-файла и записи всей информации в него (в том числе зачетной книжки студентов).

#### Решение задачи:

Класс Person. Класс, представляющий человека и содержащий методы для регистрации логов.

```
import pickle
import datetime

class Person:
    def __init__(self, first_name, last_name, age):
        # Конструктор класса Person
        self.first_name = first_name
        self.last_name = last_name
        self.age = age

    def log(self, key, comment):
        # Метод для регистрации логов
```

```

        now = datetime.datetime.now()
        log_data = f"{key} --- {now} --- {comment}"
        with open('log.txt', 'a') as f:
            f.write(log_data + '\n')

    def __str__(self):
        # Метод для получения строкового представления объекта класса
        return f"Name & Surname: {self.first_name} {self.last_name}\n\tAge: {self.age}\n"

```

Класс Student. Класс, представляющий студента и наследующий класс Person. Содержит методы для добавления оценок, получения оценок и вывода информации о студенте.

```

import pickle
import datetime
from person import Person

class Student(Person):
    def __init__(self, first_name, last_name, age, student_id, grades):
        # Конструктор класса Student
        super().__init__(first_name, last_name, age)
        self.student_id = student_id
        self.grades = grades
        self.log("CRE", f"Student created with the following name: {self.first_name} {self.last_name}")

    def log(self, key, comment):
        # Метод для регистрации логов
        now = datetime.datetime.now()
        log_data = f"{key} --- {now} --- {comment}"
        with open('log.txt', 'a') as f:
            f.write(log_data + '\n')

    def add_grade(self, subject, grade):
        # Метод для добавления оценки
        self.log("INF", f"{grade} grade has been added to {self.first_name}'s grade in subject {self.grades[subject]}")
        self.grades[subject] = grade

    def get_grade(self, subject):
        # Метод для получения оценки по предмету
        self.log("INF", f"{self.first_name}'s marks from {subject} has been printed")
        return self.grades.get(subject, None)

    def print_grades(self):
        # Метод для вывода оценок студента
        self.log("INF", f"{self.first_name}'s grades has been printed")
        for subject, grade in self.grades.items():
            print(f"{subject}: {grade}")

    def __str__(self):
        # Метод для получения строкового представления объекта класса
        self.log("INF", f"information about student: {self.first_name} {self.last_name} printed")
        return f"{super().__str__()} \tStudent ID: {self.student_id} \n\tGrades: {self.grades}"

```



Класс Professor. Класс, представляющий профессора и наследующий класс Person. Содержит методы для добавления и удаления курсов, установки и изменения должности профессора, вывода информации о профессоре и вывода списка курсов.

```
import pickle
import datetime
from person import Person

class Professor(Person):
    def __init__(self, first_name, last_name, age, professor_id, position):
        # Конструктор класса Professor
        super().__init__(first_name, last_name, age)
        self.professor_id = professor_id
        self.position = position
        self.courses = []
        self.log("CRE", f"Professor created with the following name:
{self.first_name} {self.last_name}")

    def add_course(self, course):
        # Метод для добавления курса
        self.log("INF", f"{course} course has been added to professor
{self.last_name}")
        self.courses.append(course)

    def remove_course(self, course):
        # Метод для удаления курса
        self.log("INF", f"{course} course has been removed from professor
{self.last_name}")
        if course in self.courses:
            self.courses.remove(course)

    def log(self, key, comment):
        # Метод для регистрации логов
        now = datetime.datetime.now()
        log_data = f"{key} --- {now} --- {comment}"
        with open('log.txt', 'a') as f:
            f.write(log_data + '\n')

    def print_courses(self):
        # Метод для вывода списка курсов
        self.log("INF", f"Course of professor {self.last_name} has been printed")
        if self.courses:
            print("Available courses:")
            for i, course in enumerate(self.courses, start=1):
                print(f"| {i}: {course}", end=' ')
            print('\n')
        else:
            print("No courses available!")

    def set_position(self, position):
        # Метод для установки должности профессора
        self.log("INF", f"professor {self.last_name} has been set as
{self.position}")
        self.position = position

    def change_position(self, new_position):
```

```

        # Метод для изменения должности профессора
        self.log("INF", f"professor {self.last_name} has been changed to {self.position}")
        self.position = new_position

    def __str__(self):
        # Метод для получения строкового представления объекта класса
        self.log("INF", f"information about professor: {self.first_name} {self.last_name} printed")
        return f"{super().__str__()} \tProfessor ID: {self.professor_id} \n \tPosition: {self.position}"

```

Класс Group. Класс, представляющий группу студентов и профессора. Содержит методы для добавления и удаления студентов, получения информации о человеке в группе, установки человека в группе и вывода информации о группе.

```

import pickle
import datetime
from student import Student
from professor import Professor

class Group:
    def __init__(self, group_num, students, professor):
        # Конструктор класса Group
        self.group_num = group_num
        self.students = students
        self.professor = professor
        self.log("CRE", f"group: {self.group_num} created")

    def log(self, key, comment):
        # Метод для регистрации логов
        now = datetime.datetime.now()
        log_data = f"{key} --- {now} --- {comment}"
        with open('log.txt', 'a') as f:
            f.write(log_data + '\n')

    def __len__(self):
        # Метод для получения длины группы (количества студентов)
        self.log("INF", f"{self.group_num} length has been called")
        return len(self.students)

    def get_person(self, index):
        # Метод для получения информации о человеке в группе по индексу
        if index == 0:
            self.log("INF", f"From {self.group_num} professor info has been called")
            return print(f"{index}: {self.professor}")
        elif index > 0 and index <= len(self.students):
            self.log("INF", f"From {self.group_num} student info has been called")
            return print(f"{index}: {self.students[index - 1]}")
        else:
            raise IndexError("Invalid index")

    def set_person(self, index, person):
        # Метод для установки человека в группу по индексу
        if index == 0:
            self.log("INF", f"To {self.group_num} professor {person} has been set")

```

```

        self.professor = person
    elif index > 0 and index <= len(self.students):
        self.log("INF", f"To {self.group_num} student {person} has been set")
        self.students[index - 1] = person
    else:
        raise IndexError("Invalid index")

    def remove_person(self, index):
        # Метод для удаления человека из группы по индексу
        if index == 0:
            self.log("INF", f"From {self.group_num} professor has been removed")
            self.professor = None
        elif index > 0 and index <= len(self.students):
            self.log("INF", f"From {self.group_num} student {self.students[index - 1]} has been removed")
            del self.students[index - 1]
        else:
            raise IndexError("Invalid index")

    def add_student(self, student):
        # Метод для добавления студента в группу
        self.students.append(student)
        self.log("INF", f"student : {student} has been added to the group {self.group_num}")

    def remove_student(self, student):
        # Метод для удаления студента из группы
        self.log("INF", f"student : {student} has been removed from the group {self.group_num}")
        self.students.remove(student)

    def __str__(self):
        # Метод для получения строкового представления объекта класса
        output = f"Group {self.group_num}:\n"
        output += f"Professor: {self.professor}\n"
        output += "Students:\n"
        for i, student in enumerate(self.students, start=1):
            output += f"{i}->\t {student}\n"
        self.log("INF", f"information about group: {self.group_num} printed")
        return output

```

В данном коде представлена программа, которая позволяет пользователю создавать группы, добавлять студентов и профессоров в эти группы, отображать информацию о группах и сохранять ее в файл или в файл pickle.

```

import pickle
import datetime
from person import Person
from student import Student
from professor import Professor
from group import Group

def create_student():
    print("Creating a new student...")
    first_name = input("Enter first name: ")

```

```

last_name = input("Enter last name: ")
age = int(input("Enter age: "))
student_id = input("Enter student ID: ")
grades = {}
while True:
    subject = input("Enter subject (or leave blank to finish): ")
    if not subject:
        break
    grade = int(input("Enter grade: "))
    grades[subject] = grade
new_stu = Student(first_name, last_name, age, student_id, grades)
# object_list.append(new_stu)
return new_stu

def create_professor():
    print("Creating a new professor...")
    first_name = input("Enter first name: ")
    last_name = input("Enter last name: ")
    age = int(input("Enter age: "))
    professor_id = input("Enter professor ID: ")
    position = input("Enter position: ")
    new_prof = Professor(first_name, last_name, age, professor_id, position)
    # object_list.append(new_prof)
    return new_prof

def save_groups_to_file(groups):
    with open("groups.txt", "w") as f:
        for group in groups:
            f.write(str(group))
    print ("Groups are saved to file (groups.txt)!")

def save_groups_to_pickle(object_list):
    with open("groups.pickle", "wb") as f:
        pickle.dump(object_list, f)
    print ("Groups are saved to pickle (groups.pickle)!")

def read_group_from_pickle():
    with open("groups.pickle", "rb") as f:
        object_list_readed = pickle.load(f)
    print ("Groups are read from pickle (groups.pickle)!")
    print("Available groups:")
    for group in object_list_readed:
        print(group)
    try:
        choice = int(input("Enter group number to view details (or leave blank to
continue): "))
        if (choice > 0) and (choice <=len(object_list_readed)):
            group_num = int(choice)-1
            group = object_list_readed[group_num]
            while True:
                index = input("Enter person index to view details (or leave blank to
go back): ")
                if not index:
                    break
                index = int(index)
                try:
                    group.get_person(index)
                except IndexError as e:
                    print(e)
                else:
                    action = input("Enter 'u' to update this person, 'r' to remove
this person, or leave blank to continue: ")

```

```

        if action.lower() == 'u':
            if index == 0:
                person = create_professor()
            else:
                person = create_student()
            group.set_person(index, person)
        elif action.lower() == 'r':
            group.remove_person(index)
    else:
        print ("invalid group no.")
    with open("groups.pickle", "wb") as f:
        pickle.dump(object_list_readed, f)
except ValueError:
    print ("invalid group no.")

object_list = []
def create_group():
    print("Creating a new group...")
    group_num = input("Enter group number: ")
    professor = create_professor()
    students = []
    while True:
        student = create_student()
        students.append(student)
        choice = input("Add another student? (y/n): ")
        if choice.lower() == 'n':
            break
    new_group = Group(group_num, students, professor)
    object_list.append(new_group)
    return new_group

def display_groups(groups):
    print("Available groups:")
    for group in groups:
        print(group)
    # choice = int(input("Enter group number to view details (or leave blank to
    continue): "))
    try:
        choice = int(input("Enter group number to view details (or leave blank to
    continue): "))
        if (choice > 0) and (choice <=len(groups)):
            group_num = int(choice)-1
            group = groups[group_num - 1]
            while True:
                index = input("Enter person index to view details (or leave blank to
    go back): ")
                if not index:
                    break
                index = int(index)
                try:
                    group.get_person(index)
                except IndexError as e:
                    print(e)
                else:
                    action = input("Enter 'u' to update this person, 'r' to remove
    this person, or leave blank to continue: ")
                    if action.lower() == 'u':
                        if index == 0:
                            person = create_professor()
                        else:
                            person = create_student()

```

```

        group.set_person(index, person)
    elif action.lower() == 'r':
        group.remove_person(index)
    else:
        print ("invalid group no.")
except ValueError:
    print("Please enter a valid integer choice.")
def main():
    groups = []
    while True:
        print("Choose an option:")
        print("1. Create a new group")
        print("2. Display available groups and their details")
        print ("3. Save the information to a file")
        print("4. Save the groups to a pickle file")
        print("5. Read the groups from the pickle file")
        print("6. Quit")
        choice = input("Enter choice: ")
        if choice == '1':
            group = create_group()
            groups.append(group)
        elif choice == '2':
            display_groups(groups)
        elif choice == '3':
            save_groups_to_file(groups)
        elif choice == '4':
            save_groups_to_pickle(object_list)
        elif choice == '5':
            read_group_from_pickle()
        elif choice == '6':
            break
        else:
            print("Invalid choice. Try again.")
if __name__ == '__main__':
    main()

```

#### 4) Тема «Стандартные библиотеки языка программирования»

##### Задача 1. Модули PICKLE и DATETIME

Выполняется на основе выполненного задания третьей темы.

– Необходимо создать текстовый файл и реализовать функцию логирования (без использования модуля logging). Функция должна вызываться из каждого метода ранее реализованных классов и записывать в файл строки следующего содержания: КЛЮЧ --- ДАТА И ВРЕМЯ --- КОММЕНТАРИЙ. Ключи: CRE (создание экземпляра класса), INF



(изменение), ERR (сработало исключение). Комментарий: создано ..., удален ..., добавлен ..., распечатан ...

– Создать заполненные экземпляры реализованных класса, сериализовать их. В другом питоновском файле импортировать файл с описанием класса и десериализовать объекты. Применить к десериализованным объектам различные методы.

#### Решение задачи:

Класс Person. Класс, представляющий человека и содержащий методы для регистрации логов.

```
import pickle
import datetime

class Person:
    def __init__(self, first_name, last_name, age):
        # Конструктор класса Person
        self.first_name = first_name
        self.last_name = last_name
        self.age = age

    def log(self, key, comment):
        # Метод для регистрации логов
        now = datetime.datetime.now()
        log_data = f"{key} --- {now} --- {comment}"
        with open('log.txt', 'a') as f:
            f.write(log_data + '\n')

    def __str__(self):
        # Метод для получения строкового представления объекта класса
        return f"Name & Surname: {self.first_name} {self.last_name}\n\tAge: {self.age}\n"
```

Класс Student. Класс, представляющий студента и наследующий класс Person. Содержит методы для добавления оценок, получения оценок и вывода информации о студенте.

```
import pickle
import datetime
from person import Person

class Student(Person):
    def __init__(self, first_name, last_name, age, student_id, grades):
        # Конструктор класса Student
        super().__init__(first_name, last_name, age)
        self.student_id = student_id
        self.grades = grades
        self.log("CRE", f"Student created with the following name: {self.first_name} {self.last_name}")

    def log(self, key, comment):
```

```

        # Метод для регистрации логов
        now = datetime.datetime.now()
        log_data = f"{key} --- {now} --- {comment}"
        with open('log.txt', 'a') as f:
            f.write(log_data + '\n')

    def add_grade(self, subject, grade):
        # Метод для добавления оценки
        self.log("INF", f"{grade} grade has been added to {self.first_name}'s grade
in subject {self.grades[subject]}")
        self.grades[subject] = grade

    def get_grade(self, subject):
        # Метод для получения оценки по предмету
        self.log("INF", f"{self.first_name}'s marks from {subject} has been printed")
        return self.grades.get(subject, None)

    def print_grades(self):
        # Метод для вывода оценок студента
        self.log("INF", f"{self.first_name}'s grades has been printed")
        for subject, grade in self.grades.items():
            print(f"{subject}: {grade}")

    def __str__(self):
        # Метод для получения строкового представления объекта класса
        self.log("INF", f"information about student: {self.first_name}
{self.last_name} printed")
        return f"{super().__str__()}\\tStudent ID: {self.student_id}\\n\\tGrades:
{self.grades}"

```

Класс Professor. Класс, представляющий профессора и наследующий класс Person. Содержит методы для добавления и удаления курсов, установки и изменения должности профессора, вывода информации о профессоре и вывода списка курсов.

```

import pickle
import datetime
from person import Person

class Professor(Person):
    def __init__(self, first_name, last_name, age, professor_id, position):
        # Конструктор класса Professor
        super().__init__(first_name, last_name, age)
        self.professor_id = professor_id
        self.position = position
        self.courses = []
        self.log("CRE", f"Professor created with the following name:
{self.first_name} {self.last_name}")

    def add_course(self, course):
        # Метод для добавления курса
        self.log("INF", f"{course} course has been added to professor
{self.last_name}")
        self.courses.append(course)

    def remove_course(self, course):
        # Метод для удаления курса

```

```

        self.log("INF", f"{course} course has been removed from professor
{self.last_name}")
        if course in self.courses:
            self.courses.remove(course)

    def log(self, key, comment):
        # Метод для регистрации логов
        now = datetime.datetime.now()
        log_data = f"{key} --- {now} --- {comment}"
        with open('log.txt', 'a') as f:
            f.write(log_data + '\n')

    def print_courses(self):
        # Метод для вывода списка курсов
        self.log("INF", f"Course of professor {self.last_name} has been printed")
        if self.courses:
            print("Available courses:")
            for i, course in enumerate(self.courses, start=1):
                print(f"| {i}: {course}", end=' ')
            print('\n')
        else:
            print("No courses available!")

    def set_position(self, position):
        # Метод для установки должности профессора
        self.log("INF", f"professor {self.last_name} has been set as
{self.position}")
        self.position = position

    def change_position(self, new_position):
        # Метод для изменения должности профессора
        self.log("INF", f"professor {self.last_name} has been changed to
{self.position}")
        self.position = new_position

    def __str__(self):
        # Метод для получения строкового представления объекта класса
        self.log("INF", f"information about professor: {self.first_name}
{self.last_name} printed")
        return f"{super().__str__()} \tProfessor ID: {self.professor_id} \n \tPosition:
{self.position}"

```

Класс Group. Класс, представляющий группу студентов и профессора. Содержит методы для добавления и удаления студентов, получения информации о человеке в группе, установки человека в группе и вывода информации о группе.

```

import pickle
import datetime
from student import Student
from professor import Professor

class Group:
    def __init__(self, group_num, students, professor):
        # Конструктор класса Group
        self.group_num = group_num

```

```

self.students = students
self.professor = professor
self.log("CRE", f"group: {self.group_num} created")

def log(self, key, comment):
    # Метод для регистрации логов
    now = datetime.datetime.now()
    log_data = f"{key} --- {now} --- {comment}"
    with open('log.txt', 'a') as f:
        f.write(log_data + '\n')

def __len__(self):
    # Метод для получения длины группы (количества студентов)
    self.log("INF", f"{self.group_num} length has been called")
    return len(self.students)

def get_person(self, index):
    # Метод для получения информации о человеке в группе по индексу
    if index == 0:
        self.log("INF", f"From {self.group_num} professor info has been called")
        return print(f"{index}: {self.professor}")
    elif index > 0 and index <= len(self.students):
        self.log("INF", f"From {self.group_num} student info has been called")
        return print(f"{index}: {self.students[index - 1]}")
    else:
        raise IndexError("Invalid index")

def set_person(self, index, person):
    # Метод для установки человека в группу по индексу
    if index == 0:
        self.log("INF", f"To {self.group_num} professor {person} has been set")
        self.professor = person
    elif index > 0 and index <= len(self.students):
        self.log("INF", f"To {self.group_num} student {person} has been set")
        self.students[index - 1] = person
    else:
        raise IndexError("Invalid index")

def remove_person(self, index):
    # Метод для удаления человека из группы по индексу
    if index == 0:
        self.log("INF", f"From {self.group_num} professor has been removed")
        self.professor = None
    elif index > 0 and index <= len(self.students):
        self.log("INF", f"From {self.group_num} student {self.students[index -
1]} has been removed")
        del self.students[index - 1]
    else:
        raise IndexError("Invalid index")

def add_student(self, student):
    # Метод для добавления студента в группу
    self.students.append(student)
    self.log("INF", f"student : {student} has been added to the group
{self.group_num}")

def remove_student(self, student):
    # Метод для удаления студента из группы
    self.log("INF", f"student : {student} has been removed from the group
{self.group_num}")
    self.students.remove(student)

```

```

def __str__(self):
    # Метод для получения строкового представления объекта класса
    output = f"Group {self.group_num}:\n"
    output += f"Professor: {self.professor}\n"
    output += "Students:\n"
    for i, student in enumerate(self.students, start=1):
        output += f"{i}->\t {student}\n"
    self.log("INF", f"information about group: {self.group_num} printed")
    return output

```

В данном коде представлена программа, которая позволяет пользователю создавать группы, добавлять студентов и профессоров в эти группы, отображать информацию о группах и сохранять ее в файл или в файл pickle.

```

import pickle
import datetime
from person import Person
from student import Student
from professor import Professor
from group import Group

def create_student():
    print("Creating a new student...")
    first_name = input("Enter first name: ")
    last_name = input("Enter last name: ")
    age = int(input("Enter age: "))
    student_id = input("Enter student ID: ")
    grades = {}
    while True:
        subject = input("Enter subject (or leave blank to finish): ")
        if not subject:
            break
        grade = int(input("Enter grade: "))
        grades[subject] = grade
    new_stu = Student(first_name, last_name, age, student_id, grades)
    # object_list.append(new_stu)
    return new_stu

def create_professor():
    print("Creating a new professor...")
    first_name = input("Enter first name: ")
    last_name = input("Enter last name: ")
    age = int(input("Enter age: "))
    professor_id = input("Enter professor ID: ")
    position = input("Enter position: ")
    new_prof = Professor(first_name, last_name, age, professor_id, position)
    # object_list.append(new_prof)
    return new_prof

def save_groups_to_file(groups):
    with open("groups.txt", "w") as f:
        for group in groups:
            f.write(str(group))
    print ("Groups are saved to file (groups.txt)!")

```

```

def save_groups_to_pickle(object_list):
    with open("groups.pickle", "wb") as f:
        pickle.dump(object_list, f)
    print ("Groups are saved to pickle (groups.pickle)!")

def read_group_from_pickle():
    with open("groups.pickle", "rb") as f:
        object_list_readed = pickle.load(f)
    print ("Groups are read from pickle (groups.pickle)!")
    print("Available groups:")
    for group in object_list_readed:
        print(group)
    try:
        choice = int(input("Enter group number to view details (or leave blank to
continue): "))
        if (choice > 0) and (choice <=len(object_list_readed)):
            group_num = int(choice)-1
            group = object_list_readed[group_num]
            while True:
                index = input("Enter person index to view details (or leave blank to
go back): ")
                if not index:
                    break
                index = int(index)
                try:
                    group.get_person(index)
                except IndexError as e:
                    print(e)
                else:
                    action = input("Enter 'u' to update this person, 'r' to remove
this person, or leave blank to continue: ")
                    if action.lower() == 'u':
                        if index == 0:
                            person = create_professor()
                        else:
                            person = create_student()
                        group.set_person(index, person)
                    elif action.lower() == 'r':
                        group.remove_person(index)
                else:
                    print ("invalid group no.")
                    with open("groups.pickle", "wb") as f:
                        pickle.dump(object_list_readed, f)
            except ValueError:
                print ("invalid group no.")

object_list = []
def create_group():
    print("Creating a new group...")
    group_num = input("Enter group number: ")
    professor = create_professor()
    students = []
    while True:
        student = create_student()
        students.append(student)
        choice = input("Add another student? (y/n): ")
        if choice.lower() == 'n':
            break
    new_group = Group(group_num, students, professor)
    object_list.append(new_group)
    return new_group

```



```

def display_groups(groups):
    print("Available groups:")
    for group in groups:
        print(group)
    # choice = int(input("Enter group number to view details (or leave blank to
continue): "))
    try:
        choice = int(input("Enter group number to view details (or leave blank to
continue): "))
        if (choice > 0) and (choice <=len(groups)):
            group_num = int(choice)-1
            group = groups[group_num - 1]
            while True:
                index = input("Enter person index to view details (or leave blank to
go back): ")
                if not index:
                    break
                index = int(index)
                try:
                    group.get_person(index)
                except IndexError as e:
                    print(e)
                else:
                    action = input("Enter 'u' to update this person, 'r' to remove
this person, or leave blank to continue: ")
                    if action.lower() == 'u':
                        if index == 0:
                            person = create_professor()
                        else:
                            person = create_student()
                        group.set_person(index, person)
                    elif action.lower() == 'r':
                        group.remove_person(index)
            else:
                print ("invalid group no.")
        except ValueError:
            print("Please enter a valid integer choice.")
def main():
    groups = []
    while True:
        print("Choose an option:")
        print("1. Create a new group")
        print("2. Display available groups and their details")
        print ("3. Save the information to a file")
        print("4. Save the groups to a pickle file")
        print("5. Read the groups from the pickle file")
        print("6. Quit")
        choice = input("Enter choice: ")
        if choice == '1':
            group = create_group()
            groups.append(group)
        elif choice == '2':
            display_groups(groups)
        elif choice == '3':
            save_groups_to_file(groups)
        elif choice == '4':
            save_groups_to_pickle(object_list)
        elif choice == '5':
            read_group_from_pickle()
        elif choice == '6':
            break

```

```

        else:
            print("Invalid choice. Try again.")
if __name__ == '__main__':
    main()

```

## Задача 2

Условие задачи: Реализовать две функции, вычисляющие математические формулы (файл math\_task\_X.png). Параметры формул являются аргументами функций.

### Решение задачи:

```

import math as m

def calc_x(z, k, a):
    x = (m.sqrt(abs((z**3)+ 0.25 * (k**2))))/ (0.5 + 2 * m.e**(z+4)) + (k**(1/3))/ 4
    - a * m.log(27, 3)
    return x

def calc_b(k, z):
    b = (m.sin(k*z)**2) + (1/2*m.pi) * (m.atan(2*k)**2)
    return b

while True:
    print ("1_task: enter the required args to solve the given equation: \n")
    z1 = int (input("z: "))
    k1 = int (input("k: "))
    a1 = int (input("a: "))
    print(calc_x(z1, k1, z1))

    print ("2_task: enter the required args to solve the given equation: \n")
    k2 = int (input("k: "))
    z2 = int (input("z: "))
    print(calc_b(k2, z2))

```

### Тестирование:

№	Тест	Ожидаемое значение	Полученное значение
1	z1 = 2 k1 = 3 a1 = 1	-5.635472125249789	-5.635472125249789
2	z2 = -1 k2 = 5	4.319079303774558	4.319079303774558

### Задача 3

Условие задачи: Регистрационные знаки транспортных средств В России применяются регистрационные знаки, состоящие из цифр и букв. Причём используются только 12 букв кириллицы, имеющие графические аналоги в латинском алфавите — А, В, Е, К, М, Н, О, Р, С, Т, У и Х. У частных легковых автомобилей номера — это буква, три цифры, две буквы, затем две или три цифры с кодом региона. Вам потребуется определить, является ли последовательность букв корректным номером.

#### Решение задачи:

```
import re

while True:
    user = input("enter the plate number: ")
    pattern = r"^[АВЕКМНОРСТУХ]{1}\d{3}[АВЕКМНОРСТУХ]{2}\d{2,3}$"
    match = re.match(pattern, user)
    if match :
        print("Valid license plate number")
    else:
        print("Invalid license plate number")
```

#### Тестирование:

№	Тест	Ожидаемое значение	Полученное значение
1	C227HA777	True	True
2	T22B7477	False	False
3	P290CA99	True	True
4	M227K19Y9	False	False
5	C227HA777	False	False

### Задача 4

Условие задачи: Различные модули для работы с платформой и операционной системой

– Собрать в папке файлы «task\_\*\*\*\*.py» – все ранее решенные задачи из тем А, В.

– Написать функцию, которая создаст папку «Ознакомительная папка» с двумя подпапками («тема А», «тема В»), переместит все файлы в правильные подпапки.

– Написать функцию, которая получает адрес ранее созданной папки «Ознакомительная папка» и выполнит обход всех подпапок и:

- чтение всех «task\_\*\*\*\*.py» файлов, нахождение в тексте названия функции и параметров
- программный запуск и выполнение данных файлов, подсчет времени выполнения

Решение задачи:

```
import os
import importlib.util
import time

def move_files():
    # Create the directory on the desktop
    desktop_path = os.path.join(os.path.expanduser("~"), "Desktop")
    target_folder = os.path.join(desktop_path, "oznakomitelnaya papka")
    if not os.path.exists(target_folder):
        os.mkdir(target_folder)
    # Get the list of files in the "zadania" directory on the desktop
    # global zadania_path
    zadania_path = os.path.join(desktop_path, "zadania")
    files = os.listdir(zadania_path)
    print (files)
    # Determine which theme folder to move each file to
    for file_name in files:
        if "Тема А" in file_name:
            # move the file to the "Theme_A" folder
            os.replace(os.path.join(zadania_path, file_name),
os.path.join(target_folder, "Тема А"))
        elif "Тема Б" in file_name:
            # move the file to the "Theme_B" folder
            os.replace(os.path.join(zadania_path, file_name),
os.path.join(target_folder, "Тема Б"))
    print ('done')

# move_files()

def execute_files():
    desktop_path = os.path.join(os.path.expanduser("~"), "Desktop")
    folder_path = os.path.join(desktop_path, "oznakomitelnaya papka")
    subfolders = [f.path for f in os.scandir(folder_path) if f.is_dir()]
    for subfolder in subfolders:
        files = os.listdir(subfolder)
        py_files = [file for file in files if file.startswith("task_") and
file.endswith(".py")]
```

```

for py_file in py_files:
    with open(os.path.join(subfolder, py_file), 'r', encoding='utf-8') as f:
        file_contents = f.read()
        # Find function names and parameters in the file contents
        function_name = None
        parameters = []
        lines = file_contents.split('\n')
        for line in lines:
            if line.startswith("def ") and "(" in line and line.endswith(":"):
                function_line = line.strip().split(" ")
                function_name = function_line[1].split("(")[0]
                parameters_line = line[line.index("(") + 1:line.index(")")]
                parameters = parameters_line.split(",")
        # Execute the file and measure execution time
        spec = importlib.util.spec_from_file_location(py_file[:-3],
os.path.join(subfolder, py_file))
        module = importlib.util.module_from_spec(spec)
        start_time = time.time()
        spec.loader.exec_module(module)
        end_time = time.time()
        execution_time = end_time - start_time
        # Print function name, parameters, and execution time
        print(f"File: {py_file}")
        print(f"Function: {function_name}")
        print(f"Parameters: {parameters}")
        print(f"Execution time: {execution_time} seconds")
        print("\n")

execute_files()

```

## 5) Тема «Графический интерфейс и внешние библиотеки»

Условие задачи: Разработать приложение с графическим интерфейсом, состоящее из двух вкладок:

- калькулятор валют согласно актуальному курсу (ЦБ РФ),
- динамика изменения курса валюты по отношению к рублю за указанный период.

Решение задачи: В предоставленном вами коде использованы следующие библиотеки и модули:

- Модуль locale: Предоставляет возможность работы с локализацией и форматированием чисел, дат и времени с учетом настроек локали.
- Модуль tkinter: Предоставляет инструменты для создания графического интерфейса пользователя (GUI).
- Модуль tkinter.ttk: Предоставляет дополнительные виджеты для использования в GUI.

- Модуль `datetime`: Предоставляет классы для работы с датой и временем.
- Модуль `matplotlib`: Библиотека для создания графиков и визуализации данных.
- Модуль `matplotlib.pyplot`: Предоставляет функции для создания графических изображений и графиков.
- Модуль `urllib.request`: Предоставляет функции для работы с URL-адресами и выполнения сетевых запросов.
- Модуль `xml.dom.minidom`: Предоставляет инструменты для работы с XML-документами в стиле DOM (Document Object Model).
- Модуль `xml.etree.ElementTree`: Предоставляет API для разбора и создания XML-деревьев.
- Модуль `re`: Предоставляет функции для работы с регулярными выражениями (регулярным поиском и заменой текста).

```
import locale
from tkinter import *
import tkinter.ttk as ttk
import datetime
import matplotlib
import matplotlib.pyplot as plt
import urllib.request
import xml.dom.minidom
import xml.etree.ElementTree as ET
import re

class Window:
    def __init__(self):
        self.window = Tk()
        self.window.title("Currency converter")
        self.width = "600"
        self.length = "250"
        self.window.resizable(False, False)
        self.tab_control = ttk.Notebook(self.window) #this changes the tabs
        self.tab_control.bind("<<NotebookTabChanged>>", self.change_resolution) #to
        resize the tabs

        def change_resolution(self, event):
            current_tab = self.tab_control.index("current")
            if current_tab == 0:
                self.window.geometry("600x225")
            elif current_tab == 1:
                self.window.geometry(f"{self.width}x{self.length}")

class Tab1(Window):
```

```

def __init__(self):
    super().__init__()
    self.tab = ttk.Frame(self.tab_control)
    self.tab_control.add(self.tab, text = "Currency calculator")

    #list of first currency
    self.spisok_valut1 = ttk.Combobox(self.tab)
    self.spisok_valut1.config(width=45)
    self.spisok_valut1.state(['readonly'])
    self.spisok_valut1["values"] = self.get_list_with_currency() #list of
currency
    self.spisok_valut1.current(0)
    self.spisok_valut1.grid(column=0, row=0 , padx=10,pady=10)

    #list of the second currency
    self.spisok_valut2 = ttk.Combobox(self.tab)
    self.spisok_valut2.state(['readonly'])
    self.spisok_valut2.config(width=45)
    self.spisok_valut2["values"] = self.get_list_with_currency() # сюда надо
заполнить список из валют
    self.spisok_valut2.current(4)
    self.spisok_valut2.grid(column=0, row=1 , padx=10,pady=10)

    #for inputing values
    vcmd = (self.window.register(self.validate_input), '%P')

    self.vvod_valut = Entry(self.tab, validate="key", validatecommand= vcmd)
    self.vvod_valut.grid(column=1, row=0 , padx=10,pady=10)

    #button of conversion
    btn_conversion = Button(self.tab, text="Convert", command =
self.btn_conversion_func )
    btn_conversion.grid(column = 2, row = 0, padx=10,pady=10)

    #output of the conversion
    self.label_valut = Label(self.tab , text = "Result")
    self.label_valut.grid(column = 1 , row= 1 , padx=10,pady=10)

    #for validation of the input of numbers only
    @staticmethod
    def validate_input(inp):
        pattern = r'^[0-9]*[\.]?[0-9]*$'
        if re.match(pattern, inp):
            if inp.count(".") <= 1:
                return True
            elif inp == "":
                return True
            return False
        return False

    #button of conversion
    def btn_conversion_func(self):
        value1 = str(self.spisok_valut1.get())
        value2 = str(self.spisok_valut2.get())
        #getting the value of ruble for them
        value1 = self.get_currency_val(value1)
        value2 = self.get_currency_val(value2)
        #formula for changing from one to another
        result = self.formula_currency(value1,value2, self)
        #bringing the result of the conversion
        self.change_label_valut(result,self)

    def blink(self,label , count):

```

```

        if count % 2 == 0:
            label.config(fg="black")
        else:
            label.config(fg="red")
        if count>0:
            label.after(500, self.blink, label , count-1)

    @staticmethod
    def change_label_valut(result,self):
        if isinstance(result, str):
            self.label_valut["text"] = f"{result}"
            self.blink(self.label_valut, 7)
        elif isinstance(result, int) or isinstance(result, float):
            self.label_valut["text"] = f"{result:.3f}"

    # for finding the value of ruble
    @staticmethod
    def formula_currency(value1, value2, self):
        value3 = self.vvod_valut.get()
        if not value3:
            return "Введите значение"
        else:
            result = value1 * float(value3) / value2
            return result

    #getting the latest values
    @staticmethod
    def get_list_with_currency():
        url = "http://www.cbr.ru/scripts/XML_daily.asp"
        response = urllib.request.urlopen(url)
        dom = xml.dom.minidom.parseString(response.read())
        nodeArray = dom.getElementsByTagName("Valute")
        result_list = [node.getElementsByTagName("Name")[0].childNodes[0].nodeValue
        for node in nodeArray]
        result_list.append("Russian Ruble")
        return result_list

    #according to the name of the currency, we return its value

    @staticmethod
    def get_currency_val(currency_name):
        try:
            url = f"https://www.cbr.ru/scripts/XML_daily.asp"
            with urllib.request.urlopen(url) as response:
                xml_data = response.read()
                root = ET.fromstring(xml_data)
                currency_element = None
                for elem in root.findall("./Valute"):
                    name_elem = elem.find("Name")
                    if name_elem is not None and currency_name in name_elem.text:
                        currency_element = elem
                        break

                if currency_element is None:
                    # Check if currency_name is "Russian Ruble" and return 1.0
                    if currency_name == "Russian Ruble":
                        return 1.0
                    return None
                # if currency_element is None:

                # return None

```



```

        value_elem = currency_element.find("Value")
        nominal_elem = currency_element.find("Nominal")

        if value_elem is None or nominal_elem is None:
            return None

        value = float(value_elem.text.replace(",", "."))
        nominal = float(nominal_elem.text)

        return value / nominal
    except Exception as e:
        print(f"An error occurred: {e}")
        return None

class Tab2(Tab1, Window):
    def __init__(self):
        super().__init__()
        self.tab2 = ttk.Frame(self.tab_control)
        self.tab_control.add(self.tab2, text="Currency Dynamic")
        # label of the currency
        currency_label = ttk.Label(self.tab2, text="Currency")
        currency_label.grid(column=0, row=0, padx=10, pady=10)

        # selection of currency
        self.currency_list = ttk.Combobox(self.tab2,
values=self.get_list_with_currency(), width=45)
        self.currency_list.state(['readonly'])
        self.currency_list.current(0)
        self.currency_list.grid(column=0, row=1, padx=10, pady=10)

        # button of making the graph
        build_button = ttk.Button(self.tab2, text="Make graph",
command=self.build_button_command)
        build_button.grid(column=0, row=4, padx=10, pady=10)

        # Period label
        period_label = ttk.Label(self.tab2, text="Period")
        period_label.grid(column=1, row=0, padx=10, pady=10)

        # Radiobutton week, month, quarter, year
        period_var = IntVar(value=0)
        self.period_var = period_var

        radiobutton_week = ttk.Radiobutton(self.tab2, text="Week",
variable=period_var, value=0, command=self.show_list_week)
        radiobutton_week.grab_current()
        radiobutton_week.grid(column=1, row=1, padx=10, pady=10)
        self.show_list_week()

        radiobutton_month = ttk.Radiobutton(self.tab2, text="Month",
variable=period_var, value=1, command=self.show_list_month)
        radiobutton_month.grid(column=1, row=2, padx=10, pady=10)

        radiobutton_quarter = ttk.Radiobutton(self.tab2, text="Quarter",
variable=period_var, value=2, command=self.show_list_quarter)
        radiobutton_quarter.grid(column=1, row=3, padx=10, pady=10)

        radiobutton_year = ttk.Radiobutton(self.tab2, text="Year",
variable=period_var, value=3, command=self.show_list_year)
        radiobutton_year.grid(column=1, row=4, padx=10, pady=10)

```

```

# label of choosing period
period_label = ttk.Label(self.tab2, text="Choose period")
period_label.grid(column=2, row=0, padx=10, pady=10)

# run
self.tab_control.pack(expand=True, fill=BOTH)
self.window.mainloop() # mainloop

#func of making graph
def build_button_command(self):
    value = None
    list_value = None
    handle_click = self.handle_button_click()
    if handle_click == 0:
        value = self.list_week.get()
        list_value = self.get_date_range_for_week(value)
    elif handle_click == 1:
        value = self.list_month.get()
        list_value = self.get_date_range_for_month(value)
    elif handle_click == 2:
        value = self.list_period.get()
        list_value = self.get_mondays_of_quarter(value)
    elif handle_click == 3:
        value = self.list_year.get()
        list_value = self.get_first_days_of_months(value)
    else:
        return None

    if list_value is None:
        return None

    currency_name = self.currency_list.get()
    list_value2 = [self.get_currency_for_date_and_name(date, currency_name) for
date in list_value]
    self.graph(list_value, list_value2)

def handle_button_click(self):
    selected_value = self.period_var.get()
    return selected_value

@staticmethod
def get_date_range_for_week(date_range):
    date_range = str(date_range)
    start_date, end_date = map(lambda x: datetime.datetime.strptime(x,
'%d/%m/%Y'), date_range.split())
    delta = end_date - start_date
    dates = [start_date + datetime.timedelta(days=i) for i in range(delta.days +
1)]
    return [date.strftime('%d/%m/%Y') for date in dates]

@staticmethod #method can be called without making the instance of class
def get_date_range_for_month(month_year_str):
    # преобразуем строку в объект datetime
    month_year_obj = datetime.datetime.strptime(month_year_str, "%B %Y")
    # month and year definition
    year = month_year_obj.year
    month = month_year_obj.month
    # first day of month
    first_day = datetime.date(year, month, 1)
    # last day of the month

```

```

        last_day = datetime.date(year, month, 28) + datetime.timedelta(days=4)
        last_day = last_day - datetime.timedelta(days=last_day.day)
        # making list off dates
        dates = [(first_day + datetime.timedelta(days=i)).strftime('%d/%m/%Y') for i
in range((last_day - first_day).days + 1)]
        return dates

    @staticmethod
    def get_currency_for_date_and_name(date, currency_name):
        url = "http://www.cbr.ru/scripts/XML_daily.asp?date_req={}".format(date)
        with urllib.request.urlopen(url) as response:
            tree = ET.parse(response)
            root = tree.getroot()
            for valute in root.iter('Valute'):
                name = valute.find('Name').text
                if name == currency_name:
                    value_str = valute.find('Value').text.replace(',', '.')
                    nominal_str = valute.find('Nominal').text.replace(',', '.')
                    return float(value_str) / float(nominal_str)
            return None

    def show_list_week(self):
        self.hide_list()
        self.list_week = ttk.Combobox(self.tab2, values = self.get_weekly_dates())
        self.list_week.state(['readonly'])
        self.list_week.config(width=20)
        self.list_week.grid(column=2, row=1, padx=10, pady=10)
        self.list_week.current(0)

    @staticmethod
    def get_weekly_dates():
        dates = []
        current_date = datetime.date.today()
        for i in range(12):
            # вычисляем дату начала текущей недели
            start_date = current_date -
datetime.timedelta(days=current_date.weekday())
            # вычисляем дату конца текущей недели
            end_date = start_date + datetime.timedelta(days=6)
            # добавляем даты начала и конца недели в список
            dates.append((start_date.strftime("%d/%m/%Y"),
end_date.strftime("%d/%m/%Y")))
            # переходим к предыдущей неделе
            current_date -= datetime.timedelta(weeks=1)
        # возвращаем список дат
        return dates

    def show_list_month(self):
        self.hide_list()
        self.list_month = ttk.Combobox(self.tab2, value = self.get_monthly_dates())
        self.list_month.state(['readonly'])
        self.list_month.grid(column=2, row=2, padx=10, pady=10)
        self.list_month.current(0)

    @staticmethod
    def get_monthly_dates():
        locale.setlocale(locale.LC_ALL, 'ru_RU.utf8')
        today = datetime.date.today()
        dates = []
        for i in range(24):
            date = today - datetime.timedelta(days=30*i)

```

```

        dates.append(date.strftime("%B %Y"))
    return dates

def show_list_quarter(self):
    self.hide_list()
    self.list_period = ttk.Combobox(self.tab2, values = self.get_quarter_dates())
    self.list_period.state(['readonly'])
    self.list_period.grid(column=2, row=3, padx=10, pady=10)
    self.list_period.current(0)

@staticmethod
def get_mondays_of_quarter(quarter):
    year = int(quarter.split()[-1])
    quarter_num = int(quarter.split()[0])
    if quarter_num == 1:
        start_date = datetime.datetime(year, 1, 1)
    elif quarter_num == 2:
        start_date = datetime.datetime(year, 4, 1)
    elif quarter_num == 3:
        start_date = datetime.datetime(year, 7, 1)
    elif quarter_num == 4:
        start_date = datetime.datetime(year, 10, 1)
    else:
        return []
    end_date = start_date + datetime.timedelta(days=91)
    monday_dates = []
    while start_date <= end_date:
        if start_date.weekday() == 0: # если это понедельник
            monday_dates.append(start_date.strftime('%d/%m/%Y'))
            start_date += datetime.timedelta(days=1)
    return monday_dates

@staticmethod
def get_quarter_dates():
    quarters = []
    for i in range(12):
        now = datetime.date.today() - datetime.timedelta(days=i*365/4)
        quarter = (now.month-1)//3 + 1
        quarters.append(str(quarter) + " квартал " + str(now.year))
    return quarters

def show_list_year(self):
    self.hide_list()
    self.list_year = ttk.Combobox(self.tab2, values = self.get_years_list())
    self.list_year.state(['readonly'])
    self.list_year.grid(column=2, row=4, padx=10, pady=10)
    self.list_year.current(0)

@staticmethod
def get_years_list():
    now = datetime.date.today()
    years = [now.year - i for i in range(12)]
    return years

@staticmethod
def get_first_days_of_months(year):
    date_list = []
    year = int(year)
    for month in range(1, 13):
        date = datetime.datetime(year, month, 1)
        date_list.append(date.strftime('%d/%m/%Y'))
    return date_list

```

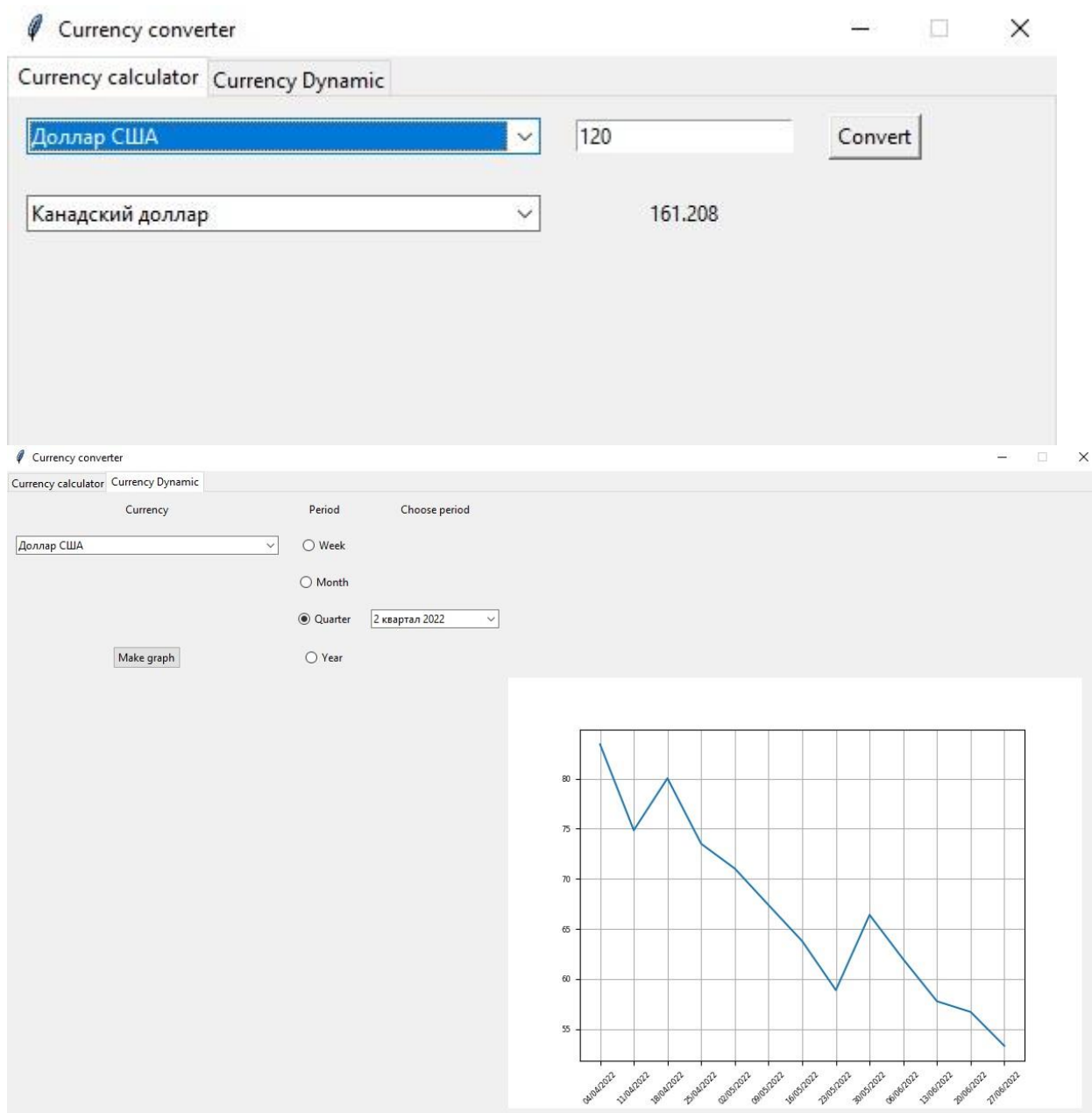
```

def hide_list(self):
    for widget in self.tab2.winfo_children():
        if isinstance(widget, ttk.Combobox):
            if widget != self.currency_list:
                widget.destroy()

def graph(self,x,y):
    plt.rc('font', size=6)
    matplotlib.use('TkAgg')
    fig = plt.figure()
    canvas = matplotlib.backends.backend_tkagg.FigureCanvasTkAgg(fig, master =
self.tab2)
    self.plot_widget = canvas.get_tk_widget()
    fig.clear()
    today = datetime.datetime.now().date() # current dates
    valid_dates = [] # valid dates that can be used
    for date_str in x:
        date = datetime.datetime.strptime(date_str, '%d/%m/%Y').date()
        if date <= today:
            valid_dates.append(date_str)
    valid_indices = [x.index(date_str) for date_str in valid_dates] # valid
dates, indexes
    valid_y = [y[i] for i in valid_indices] # values of the dates
    plt.plot(valid_dates, valid_y)
    plt.grid()
    plt.xticks(rotation=45)
    self.width = "1225"
    self.length = "725"
    self.window.geometry(f"{self.width}x{self.length}")
    self.plot_widget.grid(row=10, column=10)

if __name__ == '__main__':
    Tab2()

```



## 6) Использование сторонних API для создания приложений

-

## 7) Заключение

В результате, ознакомительная практика позволила нам овладеть основными конструкциями языка программирования, освоить работу со строками и файлами, понять принципы объектно-ориентированного программирования, изучить возможности стандартной библиотеки и

научиться создавать графический интерфейс. Полученные знания и навыки будут полезны нам в дальнейшей работе и развитии в области программирования.